Building the To Do application

for Android

Trevin Beattie

Table of Contents

1 Code Organization

As with all Android projects, the `AndroidManifest.xml` file under `app/src/main/` describes the main components of the application — its activities, provider, services, and intent receiver.

There are two major packages for the code in `app/src/main/java/`: the cryptography code and the main application. All code is under the top-level package `com.xmission.trevin.android`, with the cryptography code in the `crypto` subpackage and the rest of the application in `todo`.

Resources used by the application (UI layout, language-dependent strings, icons, etc.) are under `app/src/main/res/`. See Section 2.2 [Generating Icons], page 5, for details on how the icons are rendered.

1.1 Encryption Code

When the user sets a password for private To Do items, the application turns the password into an encryption key using a Password-Based Key Derivation Function (PBKDF (`https://en.wikipedia.org/wiki/PBKDF2`)). Android 4.4 (KitKat) introduced a **breaking change** to the SecretKeyFactory resulting in a different key being generated for the same password compared to the key generated in Android 4.3 (Jellybean) and earlier (see “Changes to the SecretKeyFactory API in Android 4.4 (`https://android-developers.googleblog.com/2013/12/changes-to-secretkeyfactory-api-in.html`)”). In addition, Android 2.2 (Froyo) had no support for the `PBKDF2WithHmacSHA1` algorithm. To allow the password to work for both local data and backup files across all supported Android versions, the app uses a copy of parts of BouncyCastle’s encryption library. This code **must** remain unchanged in all versions of the To Do app.

1.2 Data Model

To Do . . .

1.3 User Interface

The Android Activities and other UI-related classes are in the `ui` subpackage. These include:

- `ToDoListActivity` — the main entry point of the application, which shows the list of To Do items.
- `ToDoDetailsActivity` — The screen which shows the general details of a single item from the list.
- `ToDoNoteActivity` — The screen that lets you add a longer note to a To Do item.

- `CalendarDatePicker` — A custom calendar-like view for selecting a due date.
- `CalendarDatePickerDialog` — A dialog wrapper around `Calendar DatePicker`
- `RepeatEditor` — The screen that lets you customize when a repeating task needs to be done again once it has been marked complete.
- `RepeatDialog` — A dialog wrapper around `RepeatEditor`
- `CategoryListActivity` — The screen that lets you edit categories for grouping items.
- `PreferencesActivity` — The "Settings" screen of the application.
- `ExportActivity` — The screen from which you can save the app's data to a backup file.
- `ImportActivity` — The screen from which you can load app date from a backup file.

1.4 Services

Background services are in the `service` subpackage. These are:

- `AlarmService` — Keeps track of which To Do items have an upcoming alarm, and posts notifications when the next one(s) are due.
- `PasswordChangeService` — When you change your password for private records, it changes their encryption key so all records need to be decrypted with the old password and re-encrypted with the new one. This service is responsible for updating the stored data. The changes occur in the following sequence:
  1. If there was an old password, all private records are decrypted.
  2. The hash of the old password (if any) is removed from the database.
  3. The hash of the new password (if any) is written to the database.
  4. If there is a new password, all private records are encrypted.

  This ensures that if the process fails at any point, you will never have data encrypted by two different keys.
- `PalmImporterService` — This service is used when importing a `.dat` file, which the app assumes was created by Palm Desktop (`https://palmdb.net/app/palm-desktop`) as a backup of a To Do list from a Palm Pilot (`https://en.wikipedia.org/wiki/PalmPilot`) or Palm Tungsten (`https://en.wikipedia.org/wiki/Palm_Tungsten`) PDA.
- `XMLExporterService` — This service is used when exporting your To Do list(s) to a backup file.
- `XMLImporterService` — This service is used when importing To Do items from an XML file. (The app currently assumes any file not ending in `.dat` is XML data.)

1.5 Receiver

In the `receiver` subpackage, the `AlarmInitReceiver` is called when Android starts up so that it can start the `AlarmService` which will let you know of any To Do items which came due while the device was off.

1.6 Provider

In the `provider` subpackage, `ToDoProvider` provides access to the SQLite database containing the To Do items and list of categories.

1.7 Utilities

The following utility classes are in the `util` subpackage.

- `StringEncryption` — This class is used to encrypt and decrypt private records, and is also the in-memory store for the password when it is entered on the Settings screen (*if* the password's hash matches the hash stored in the database.) The code takes care to ensure the plain-text password is not cached, nor is it ever written to persistent storage.

  `StringEncryption`
- `FileUtils` — This class is used to determine the default location where backup files will be written or read, and to check whether an SD card is mounted and whether the app has permission to write to or read from the requested folder.

1.8 Unit Tests

The unit tests are found in `app/src/test/java/` and is comprised of tests that can run on a standard Java platform rather than the Android platform. In some cases, parts of the Android library (e.g. the `android.util.Log` facility) have mock substitutes.

1.9 Android Tests

The tests in `app/src/androidTest/java/` *require* an Android platform to run on, typically an emulator. These should be repeated on emulators running different Android versions to check compatibility.

2 Building from the Source

When making any changes to the code, you also need to change the release date in the "About..." dialog (`res/values/strings.xml`: `InfoPopupText`) as well as the application's internal version number (`AndroidManafest.xml`: `android:versionCode` and `android:versionName`).

2.1 Development Environment

Building the application `.apk` was originally done up until 2014 using:

- A Java 1.6 compiler
- Eclipse integrated development environment (IDE)
- The Android software development kit (SDK) release 12.

To build the old code (version 1.2.x) on a newer system requires a few different tools, but not *too* modern. The following development environment was set up and tested in 2025 on Fedora Linux 40 with only minor bug fixes to the code:

- Java (1.)8 *(Do* **not** *use a newer version, as Gradle 3.5 is not compatible with Java 11 or higher.)*
- Android Studio 2.3.3, downloaded from the archive (`https://developer.android.com/studio/archive`).
- Gradle 3.5, installed by setting up the Gradle Wrapper for the project and then modifying `gradle/wrapper/gradle-wrapper.properties` to set `distributionUrl` to `https\://services.gradle.org/distributions/gradle-3.5-all.zip`.
- The Android Gradle Plugin version 2.3.3, which had to be downloaded along with most of its dependencies from a 3rd-party mirror `https://repository.axelor.com/nexus/service/rest/repository/browse/maven-public/` as it does not exist in Maven Central nor in Google's Maven repository.

The following development environment was also tested:

- Java (1.)8 *(Do* **not** *use a newer version, as Gradle 4.1 is not compatible with Java 11 or higher.)*
- Android Studio 3.0, downloaded from the archive (`https://developer.android.com/studio/archive`).
- Gradle 4.1, installed by running `./gradlew wrapper --gradle-version=4.1`.
- The Android Gradle Plugin version 3.0.0, available in Maven Central.

Building the more recent code (version 1.3+) *should* work on the latest release of Android Studio; the following environment was used to build version 1.3.0:

- Java 21

- Android Studio 2024.2.2 (“Ladybug”)
- Gradle 8.12.1

See `http://developer.android.com/sdk/index.html` for information on how to set up a project using the Android SDK.

2.2 Generating Icons

Some of the application icons are generated from 3-D image description files using Persistence of Vision (`http://www.povray.org/`). The image files are included with the source code (in `app/src/main/res/drawable*/`, but in case you want to tweak or change any image you can use `povray` to generate new ones. The `.pov` sources are under `app/src/main/graphics/`.

Because some of the raster icons contain small details that may get lost if rendered at a low resolution, it is recommended that you render the initial image at a size which is the least common multiple of all icon sizes — 288×288 — and then use a raster graphics program such as the GIMP (`http://www.gimp.org/`), `convert` (`http://www.imagemagick.org/`), or `pamscale` (`http://netpbm.sourceforge.net/`). For each image file `foo.png`, you need to create several icons, aiming for a size of about $\frac{1}{5}$ inch on the screen: for example, a 16×16 icon in `res/drawable/foo_16.png`, a 32×32 icon in `res/drawable/foo_32.png`, a 36×36 icon in `res/drawable/foo_36.png`, a 24×24 icon in `res/drawable-ldpi/foo.png`, a 32×32 icon in `res/drawable-mdpi/foo.png`, a 48×48 icon in `res/drawable-hdpi/foo.png`, and a 64×64 icon in `res/drawable-xhdpi/foo.png`.

The command for generating an image from one of the `.pov` descriptions is:

```
povray +FN +AM3 +A0.3 +UA +W288 +H288 foo.pov
```

The main application icon was drawn by hand in the GIMP, and can be found in `IconMaster.xcf`. *For Android Nougat and earlier,* this raster image is scaled down from 384×384 to about $\frac{1}{2}$ inch on screen: a 60×60 icon in `res/drawable-ldpi/icon.png`, 80×80 in `res/drawable-mdpi/icon.png`, 120×120 in `res/drawable-hdpi/icon.png`, and 160×160 in `res/drawable-xhdpi/icon.png`.

Android Oreo introduced “adaptive icons”, so the usual icon — which has the natural rectangular shape of a piece of paper — had to be squeezed down to fit into a mask that can be anything from a rounded squircle to a circle. To generate the various sizes of this icon, the canvas has to be enlarged to 588×588 and then scaled back down to 80×80, 120×120, 160×160, and 240×240 for the various pixel densities in `res/drawable-*dpi/icon_foreground.png`. In addition, a background image has to be provided in `res/drawable-*dpi/icon_background.png`.

Reproducibility Issues

Some publishing systems (e.g. F-Droid) require that compiled code and generated / modified resources be kept exactly the same across any build environment. When writing PNG images, applications tend to use the system's "zlib" compression library, which is ordinarily the one provided from zlib.net. Some systems may replace this library with an alternative such as "zlib-ng"; in order to get consistent output, you may need to install "zlib" separately and have this library loaded when running Android Studio, '`./gradlew`', and any other application you use to create PNG images. For example, the following may be used from the command line:

```
export LD_PRELOAD=/usr/local/lib/libz.so.1
```

which will cause the application to use the zlib implementation in `/usr/local/lib/` instead of the default implementation in `/usr/lib/` or `/usr/lib64/`.

EXIF Prohibition

Some publishing platforms such as F-Droid forbid any EXIF metadata in images. The `exiftool` (`https://exiftool.org/`) utility may be used to strip this metadata from your images:

```
exiftool -all= drawable*/*.png
```

The tool renames the original files to e.g. `abc.png_original`.

2.3 Testing

Currently, there are no automated tests for the source code; testing must be done by running the application either on an emulator or real Android device which has USB Debugging enabled.

2.3.1 Using Virtual Devices

If you want to test the application on different versions of Android or different types of devices than you have physical devices for (or don't want to use real devices for testing), you will need to configure emulators using the Android Virtual Device Manager.

In Android Studio 2.3, go to Tools → Android → AVD Manager to open the Android Virtual Device Manager. In Android Studio 2024, this is at Tools → Device Manager. Click on "Create Virtual Device" (or the '`+`' icon) to add a new emulator. On the first page you will select the type and screen size of the device, e.g. small phone or large tablet, along with its pixel resolution.

On the next page choose which version of Android the emulator will run. The ABI determines what type of CPU the system will run on: "arm",

"arm64", "x86", or "x86_64". In order for the emulator to run the ABI **must** match (or be compatible with) your computer's host CPU; for example, an "arm" image *will not run* on an "x86_64" computer. (This means that you cannot test on Android 2.2 (Froyo) or earlier in emulation, since there are no x86 builds for those versions of Andriod.)

The system image list shows both the Android version (e.g. 6.0), code name (e.g. Marshmallow), and API level (e.g. 23). You should create at least enough emulators to cover the minimum and target SDK versions specified in `AndroidManifest.xml`, and ideally the latest release of Android and a few intermediate versions to make sure the app is compatible across a wide range of device ages. It is *highly recommended* to test against the following API's, since the code has branches that follow different paths for each of these versions:

- Ice Cream Sandwich (Android 4.0, API 14–15), which is the current minimum version
- Jelly Bean (Android 4.1, API 16–18)
- KitKat (Android 4.4, API 19–20) through Lollipop (Android 5.1, API 22)
- Marshmallow (Android 6, API 23)
- Nougat (Android 7, API 24–25)
- Oreo (Android 8.1, API 27)
- Pie (Android 9, API 28)
- Quince Tart (Android 10, API 29) through Red Velvet Cake (Android 11, API 30)
- Snow Cone (Android 12, API 31–32), which is the current target version

Testing the following API's is suggested to ensure compatibility with new devices:

- Tiramisu (Android 13, API 33) through Upside-down Cake (Android 14, API 34)
- Vanilla Ice Cream (Android 15, API 35) and up

On the last page finalize the details of the virtual device. Give it a descriptive name which will distinguish it from other virtual devices; for example, include the type/size of the device and the Android version and/or API level. Under "Advanced Settings" you can configure the system's CPU and memory usage.

To run the virtual device for testing, click on its play button in the device list. The emulated device should start up in a new window, acting as if it were powered on.

2.3.2 Enabling USB Debugging

You will need to do this whether you are using a virtual or real device.

First, if your device runs Android 4.2 (Jelly Bean) or higher you will need to enable Developer Mode. To do this, go to the system Settings and under "About phone" look for the "Build number". (This may be nested under "Software information".) Tap on the build number **seven times**; you should see the message "You are now a developer!".

Next find the "Developer options" in the system settings and scroll down to the "USB debugging" switch. Turn that on whenever you need to test the application, and plug the phone into your computer that is running Android Studio. For security you should always turn this setting back off when you are finished testing.

2.3.3 Debugging the Application

In Android Studio (with the ToDo project open), ensure you have a valid build. If you will be testing on a virtual device be sure that the emulator is running. See Section 2.3.1 [Using Virtual Devices], page 6, for how to configure and run the virtual device.

Click the debug icon or go to Run → Debug 'app', then in the "Select Deployment Target" window select the device under "Connected Devices". If that section shows "<none>" then either the device is not running, not connected (by a USB cable if it's a physical device), doesn't have USB debugging enabled, or may have some other problem with it. When you click "OK" then Studio should install the current app build onto the device and launch it.

The "Android Monitor" panel may be used to view any log messages from the application or the Android system. Be sure the correct device is selected. When the application is started in debug mode, Android will wait for Android Studio's debugger to connect before the application will run so that it can catch any startup errors. If the application crashes, check then panel for an exception error message which should show the point in the code where the error occurred.

2.4 Generating the Application Package (APK)

Version Control

If you will be generating a signed APK for publishing, *and* you are using Gradle version 8.3 or higher, **commit all of the source code** prior to generating the APK. The Android Gradle plugin includes the commit hash in `META-INF/version-control-info.textproto`.

In Android Studio, click on the Build menu then "Build APK". If there were no errors, this should produce `app/app-release.apk` (in Android Studio 2.3) or `app/release/app-release.apk` (in Android Studio 2024). You should rename this file to a more descriptive name like `todo-1.3.0.apk`.

Alternatively from the command line, run the following commands:

```
./gradlew assembleRelease
```

This will produce an *unsigned* APK in `app/build/outputs/apk/release/app-release-unsigned.apk`. To sign this, you will need to locate the build tools for the version specified in `app/build.gradle` under `android.buildToolsVersion` (if present; otherwise the most recent build tools supported by the Android Gradle plugin). You will also need to know where the keystore of your app signing key is located.

```
${BUILD_TOOLS_VERSION_DIRECTORY}/apksigner sign \
  --alignment-preserved \
  --ks ${ANDROID_KEYSTORE} --ks-key-alias ${SIGNING_KEY_ALIAS} \
  --out app/${APP_NAME}-${VERSION}.apk \
  app/build/outputs/apk/release/app-release-unsigned.apk
```

(The `--alignment-preserved` *option is needed if you are using build tools 35 or higher. Omit it for earlier version of the build tools.)*

2.5 Generating Documentation

Lastly, if you need to generate a new edition of this manual, you will need `texinfo` (`http://www.gnu.org/software/texinfo/`) and `texi2pdf` (`https://www.gnu.org/software/texinfo/manual/texinfo/html_node/Format-with-texi2dvi-or-texi2pdf.html`) and/or `texi2html` (`http://www.nongnu.org/texi2html/`).

To generate the manual in PDF, simply run:

```
texi2pdf ToDo.texinfo
```

To generate the manual in HTML, run:

```
texi2html --split node ToDo.texinfo
```

which will produce the manual under `doc/ToDo/ToDo.html`.

3 To Do

- Document how the code works
- Document the data structures
- Document the encryption algorithms